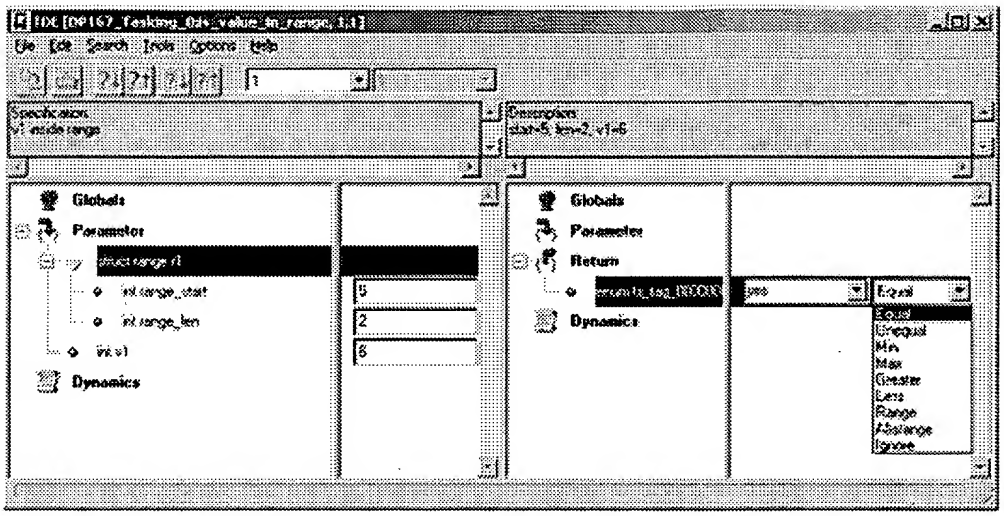

Figure 1: The **Test** Interface Editor (TIE) displays the number, type and passing direction of interface variables

Tessy then automatically **generates** additional source code for the so-called **test** driver, which calls the function to be **tested**. If necessary, Tessy can also **generate** code for placeholders, which are used as substitutes for functions or external variables that are not yet implemented. Tessy then **generates** the **test** application using the start up code of the relevant microcontroller, the **test** driver, the optional placeholders and the function under **test**. In doing so, a cross compiler for the relevant target system is usually used.

Test case input values and the values of expected **test** case results can now be specified in Tessy using its integrated **Test** Data Editor (TDE). The TDE graphically displays the interface of the function under **test** and also allows the viewing of objects ranging from complex interface elements such as structures to elementary data types. Although the inputting of **test** data is a major task for the user, Tessy's TDE allows this to be accomplished with particular ease and efficiency. The TDE can also be used to specify the expected results of a **test** case. Values are automatically stored in a database

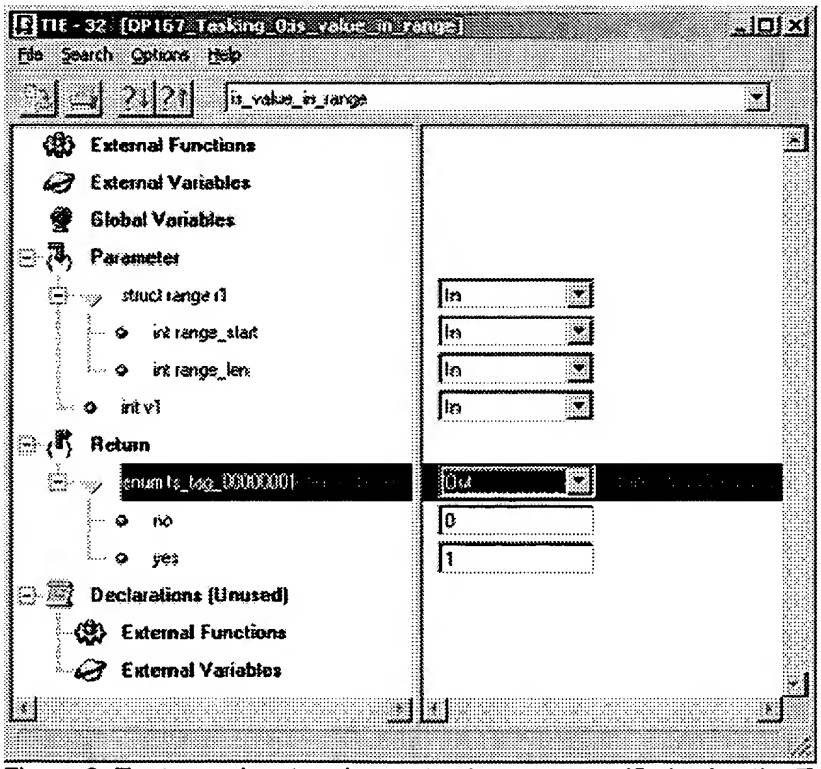

Figure 2: **Test** case input and output values are specified using the **Test** Data Editor (TDE)

In order to run a **test**, Tessy loads the **test** application via a **debugger** into the **test** system, which could be an in-circuit emulator. The **test** thus makes use of the actual microcontroller and also checks the cross compiler used. Tessy then runs all **test cases** in sequence on the **test** system and checks if the actual result of each **test** case executed corresponds with the expected result. **Test** values are extracted from the database one by one and are not included in the **test** application. This allows the size of a **test** application to be independent of the number of **test cases**. Hence in principle, the number of **test cases** is unlimited and Tessy can be used with 8-bit microcontrollers that have limited memory.

Tessy **generates** reports on the execution and results of **test cases** in different degrees of detail.

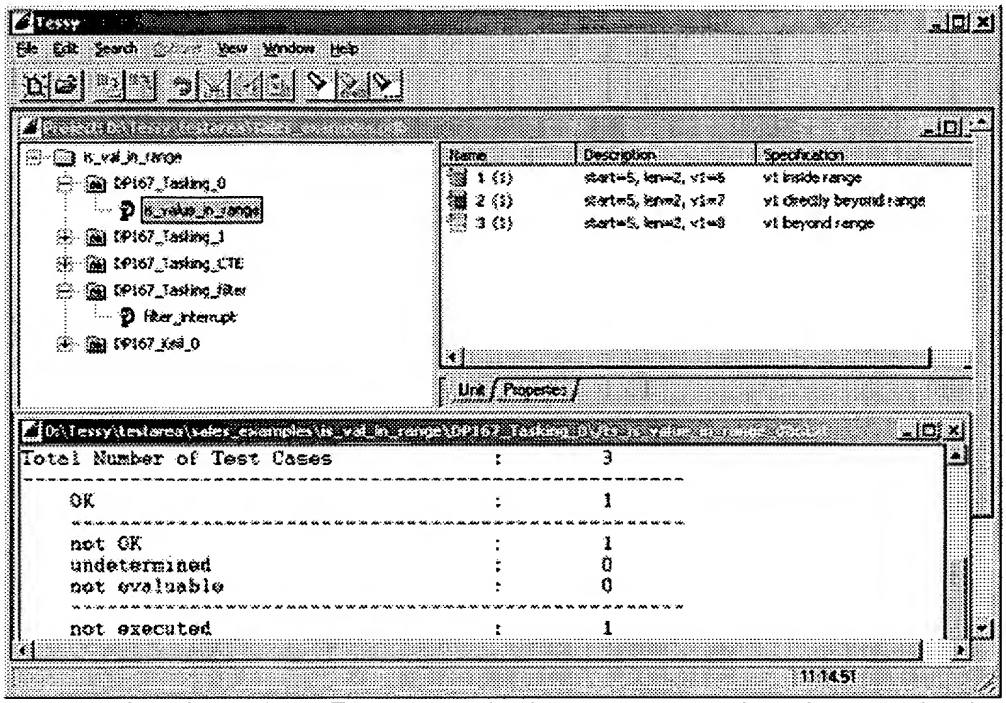

Figure 3: **Test** Evaluation – **Test cases** yielding an unexpected result are marked in red

Of course, if the execution of a **test** case with a particular set of input values did not yield the expected result, the causes of this failure need to be determined. The tight integration of Tessy with **debuggers** such as Hitex's HiTOP enable Tessy to re-run the **test** case, whereby Tessy sets a breakpoint at the entry point of the function to be **tested**. **Test** case execution is thus conveniently transferred to the **debugger** where it continues at the very beginning of the function to be **tested**. The function is called with the exact data that caused the unexpected result to occur. This allows the error detection process to begin immediately. As soon as the error is found, Tessy's integral editor can be used to correct the function's source code and all **tests** can then easily be re-run using the newly modified **test** object.

Regression **tests** are re-runs of successfully completed **test cases**. They provide verification that the modifications and enhancements made to a program are not the cause of undesired effects. Automatic regression **tests** are particularly useful after optimizing software or whenever a new compiler version is used. Tessy includes a batch function that enables extensive regression **testing** to take place without any user intervention.

Further development of a program can sometimes cause **test cases** to become un-executable, since the interface of the function under **test** has changed, for example due the introduction of an additional parameter. With self-created **test** environments, a large amount of adaptation is usually necessary. Tessy however completes this adaptation comfortably and to a large extent automatically.

In order for **test cases** to be updateable and thus reusable at any time, **test** data, including data for modified interfaces must be reusable in the most efficient and extensive means possible. The re-running of **tests** implies regression **testing** and regression **testing** is an essential software quality assurance procedure. Only with regression **tests** is it possible to optimize software in terms of size, performance, maintainability and reusability, since regression **tests** guarantee the actual functionality remains unaltered.

Test Coverage Analysis indicates the extent to which **testing** has covered the software under **test**. If a certain percentage of coverage is lacking, it could be that the **tests** were insufficient or some other abnormality exists such as "dead code."

Tessy also determines the paths of program flow covered when running a **test**. By comparing the number of paths covered to the number of paths possible, Tessy is able to reveal the Path Coverage, also known as C1 Coverage. Tessy also indicates the paths covered by individual **test cases**: if the Path Coverage was insufficient, it's easy to recognize the paths that were not **tested**.
The Classification Tree Editor (CTE) is an integral part of Tessy and it is used to specify **test cases** based on the Classification Tree Method. The Classification Tree Method is a systematic means of determining a set of low-redundancy, error sensitive **test cases** and it leads from a problem specification to actual **test** case specifications. The so-called classification tree is created during this process.

Use of the Classification Tree Method requires careful consideration of specifications and **test cases**. This allows insight to be gained into issues such as:

- What aspects are available for the **test** and how are they to be covered by **test cases**?
- What number of **test cases** is sufficient?
- Are there any redundant **test cases**?
- What is the expected amount of **testing** required?

The CTE is a syntax controlled graphical editor used to create a classification tree and to determine the **test** case specifications that are based on this tree. The CTE manages **test** case specifications, **test cases** and **test** case remarks, etc. and it can provide this information to other tools, as well as to Tessy.

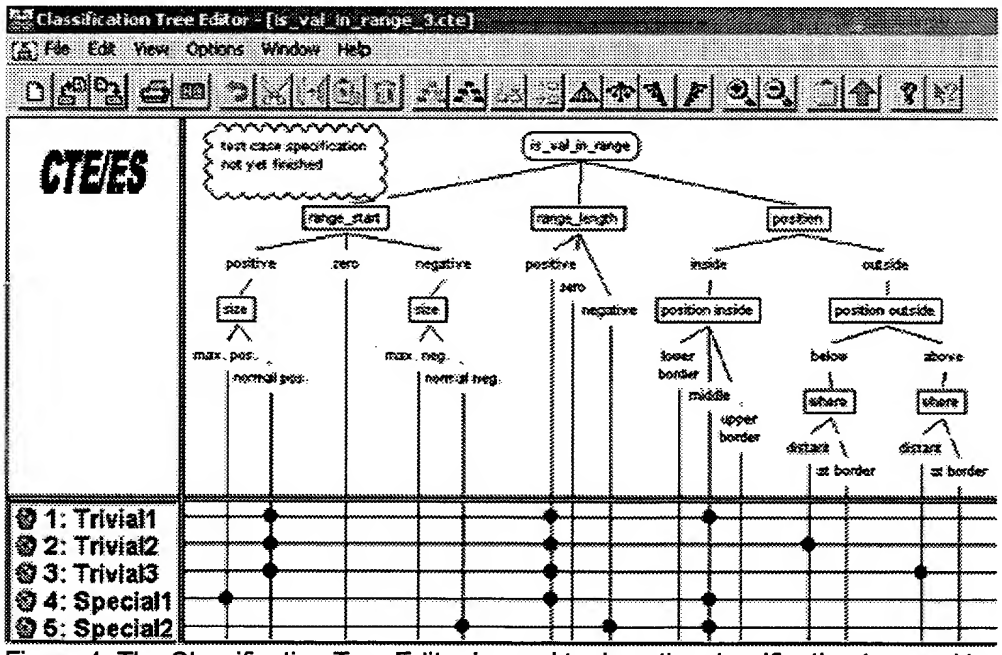

Figure 4: The Classification Tree Editor is used to draw the classification tree and to specify **test cases**

Tessy greatly improves the ease and efficiency of **testing** applications and to a large extent it replaces proprietary **test** environments. At the beginning of a **test**, Tessy **generates** any components still missing such as **test** drivers or placeholder functions. The tool's **test** data management allows all **test** data to be conserved for use in subsequent **tests**. The integration of Tessy with **debuggers** allows efficient error detection whenever **test cases** yield unexpected results. If a program is modified, Tessy can be used to correspondingly update and adapt the program's **tests**. Tessy is thus particularly suited to regression **testing**.

The planning of **test cases** at the beginning of development ties in with the **test** coverage analysis at the end of **testing**.

Tessy and the CTE are available from Hitex Development Tools.

Author:
Frank Büchner

Contact
Hitex Development Tools, Greschbachstrasse 12, D-76229 Karlsruhe / Germany
www.hitex.de / www.hitex.com

Apache HTTP Server Version 1.3

Module mod_setenvif

This module provides the ability to set environment variables based upon attributes of the request.

Status: Base
Source File: mod_setenvif.c
Module Identifier: setenvif_module
Compatibility: Available in Apache 1.3 and later.

Summary

The `mod_setenvif` module allows you to set environment variables according to whether different aspects of the request match regular expressions you specify. These environment variables can be used by other parts of the server to make decisions about actions to be taken.

The directives are considered in the order they appear in the configuration files. So more complex sequences can be used, such as this example, which sets `netscape` if the browser is mozilla but not MSIE.

```
BrowserMatch ^Mozilla netscape
BrowserMatch MSIE !netscape
```

For additional information, we provide a document on Environment Variables in Apache.

Directives

- BrowserMatch
- BrowserMatchNoCase
- SetEnvIf
- SetEnvIfNoCase

BrowserMatch directive

Syntax: BrowserMatch *regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
Default: *none*
Context: server config, virtual host, directory, .htaccess
Override: FileInfo
Status: Base

Module: mod_setenvif
Compatibility: Apache 1.2 and above (in Apache 1.2 this directive was found in the now-obsolete mod_browser module); use in .htaccess files only supported with 1.3.13 and later

The BrowserMatch directive defines environment variables based on the `User-Agent` HTTP request header field. The first argument should be a POSIX.2 extended regular expression (similar to an `egrep`-style regex). The rest of the arguments give the names of variables to set, and optionally values to which they should be set. These take the form of

1. *varname*, or
2. `!`*varname*, or
3. *varname*`=`*value*

In the first form, the value will be set to "1". The second will remove the given variable if already defined, and the third will set the variable to the value given by *value*. If a `User-Agent` string matches more than one entry, they will be merged. Entries are processed in the order in which they appear, and later entries can override earlier ones.

For example:

```
BrowserMatch ^Mozilla forms jpeg=yes browser=netscape
BrowserMatch "^Mozilla/[2-3]" tables agif frames javascript
BrowserMatch MSIE !javascript
```

Note that the regular expression string is **case-sensitive**. For case-INsensitive matching, see the `BrowserMatchNoCase` directive.

The `BrowserMatch` and `BrowserMatchNoCase` directives are special cases of the `SetEnvIf` and `SetEnvIfNoCase` directives. The following two lines have the same effect:

```
BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
```

BrowserMatchNoCase directive

Syntax: BrowserMatchNoCase *regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
Default: *none*
Context: server config, virtual host, directory, .htaccess
Override: FileInfo
Status: Base
Module: mod_setenvif
Compatibility: Apache 1.2 and above (in Apache 1.2 this directive was found in the now-obsolete mod_browser module)

The `BrowserMatchNoCase` directive is semantically identical to the `BrowserMatch` directive. However, it provides for case-insensitive matching. For example:

```
BrowserMatchNoCase mac platform=macintosh
BrowserMatchNoCase win platform=windows
```

The `BrowserMatch` and `BrowserMatchNoCase` directives are special cases of the `SetEnvIf` and `SetEnvIfNoCase` directives. The following two lines have the same effect:

```
BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
```

SetEnvIf directive

Syntax: SetEnvIf *attribute regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
Default: *none*
Context: server config, virtual host, directory, .htaccess
Override: FileInfo
Status: Base
Module: mod_setenvif
Compatibility: Apache 1.3 and above; the Request_Protocol keyword and environment-variable matching are only available with 1.3.7 and later; use in .htaccess files only supported with 1.3.13 and later

The `SetEnvIf` directive defines environment variables based on attributes of the request. These attributes can be the values of various HTTP request header fields (see RFC2616 for more information about these), or of other aspects of the request, including the following:

- `Remote_Host` - the hostname (if available) of the client making the request
- `Remote_Addr` - the IP address of the client making the request
- `Remote_User` - the authenticated username (if available)
- `Request_Method` - the name of the method being used (`GET`, `POST`, *et cetera*)
- `Request_Protocol` - the name and version of the protocol with which the request was made (*e.g.*, "HTTP/0.9", "HTTP/1.1", *etc.*)
- `Request_URI` - the portion of the URL following the scheme and host portion

Some of the more commonly used request header field names include `Host`, `User-Agent`, and `Referer`.

If the *attribute* name doesn't match any of the special keywords, nor any of the request's header field names, it is tested as the name of an environment variable in the list of those associated with the request. This allows `SetEnvIf` directives to test against the result of prior matches.

> **Only those environment variables defined by earlier `SetEnvIf[NoCase]` directives are available for testing in this manner. 'Earlier' means that they were defined at a broader scope (such as server-wide) or previously in the current directive's scope.**

Example:

```
SetEnvIf Request_URI "\.gif$" object_is_image=gif
SetEnvIf Request_URI "\.jpg$" object_is_image=jpg
```

```
SetEnvIf Request_URI "\.xbm$" object_is_image=xbm
    :
SetEnvIf Referer www\.mydomain\.com intra_site_referral
    :
SetEnvIf object_is_image xbm XBIT_PROCESSING=1
```

The first three will set the environment variable `object_is_image` if the request was for an image file, and the fourth sets `intra_site_referral` if the referring page was somewhere on the `www.mydomain.com` Web site.

SetEnvIfNoCase directive

Syntax: SetEnvIfNoCase *attribute regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
Default: *none*
Context: server config, virtual host, directory, .htaccess
Override: FileInfo
Status: Base
Module: mod_setenvif
Compatibility: Apache 1.3 and above; the Request_Protocol keyword and environment-variable matching are only available with 1.3.7 and later; use in .htaccess files only supported with 1.3.13 and later

The `SetEnvIfNoCase` is semantically identical to the `SetEnvIf` directive, and differs only in that the regular expression matching is performed in a case-insensitive manner. For example:

```
SetEnvIfNoCase Host Apache\.Org site=apache
```

This will cause the `site` environment variable to be set to "`apache`" if the HTTP request header field `Host:` was included and contained `Apache.Org`, `apache.org`, or any other combination.

Apache HTTP Server Version 1.3